Topics in Natural Language Processing Assignment

Kevinkumar Patel
Student Id: 1107736
Lakehead University

***Abstract*—Implementing an one dimensional convolution (Conv1D)-based neural network for predicting median house value using longitude, latitude, housing median age, total number of rooms, total number of bedrooms, population, number of households, and median income using the non-linear regression technique.**

I. Introduction

The prediction of median house value as they vary through time and place is a complex task, particularly when the available data is massive and divided into many different data types. Dataset is used for this model is modified California housing dataset and 70% data is used for training the model and 30% of the data is used for testing the model. I have used one dimensional convolution (Conv1D)-based neural network for predicting median house value using longitude, latitude, housing median age, total number of rooms, total number of bedrooms, population, number of households, and median income. This work is combined with non linear regression technique and one dimensional convolution(Conv1D) where Non linear regression is the form of regression analysis in which observational data are modeled by a function which is a nonlinear combination of the model parameters and depends on one or more independent variables.

II. Dataset

The dataset is used in this work is modified version of the California Housing dataset (housing.csv). first 10 rows of the dataset is following. This dataset has 20433 instances and 10 features which are 'longitude', 'latitude', 'housing median age', 'total number of rooms', 'total number of bedrooms', 'population', 'number of households', and 'median income' respectively.following is the 10 instances of this dataset.This dataset has some incomplete entries.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0	NEAR BAY
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0	NEAR BAY
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.1200	241400.0	NEAR BAY
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY
9	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0	NEAR BAY

Fig. 1. First 10 instances of the database

III. Preprocessing

To start with, These are the libraries which have been used to perform the task.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import csv
import numpy as np
```

Fig. 2. Libraries used to perform the task

First of all I got the dataset from the online url link and passed to the pandas method and also In this given housing dataset there are some missing entries found so to fill the data i have used pandas dropna() method. It fills the appropriate value on that particular place.

```
[ ]
    url = "https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.csv"

    dataset = pd.read_csv(url)
    dataset = dataset.dropna()
```

Fig. 3. code-snippet to fill the missing value

IV. Defining the model

After that with the use of the pandas library and read_csv() method I read the dataset and printed 10 instances of the dataset. Then after I created a plot for initial 18 instances and following is code snippet and result of the code.

```
ax_list = [] # used to store the object of the subplot
lst = [i for i in range(18)]
color_list = ["red","blue","grey","purple","yellow","pink","green","brown","violet","black"]
fig, ax = plt.subplots(10)
for i in range(10):
  ax_list.append(ax[i])
feature_list = []
for i in dataset:
  feature_list.append(str(i))
for i in range(len(ax_list)):
  ax_list[i].plot(lst,dataset[feature_list[i]][0:18],color_list[i])
  ax_list[i].legend([feature_list[i]])
```

Fig. 4. Code snippet to generate the plot

So below is the plot of the above code snippet.which shows the correct 18 instances of the dataset and 10 features of the dataset according to that.In above code snippet i have used matplot library to draw the dataset. which is widely used in data analysis and other machine learning related analysis also.

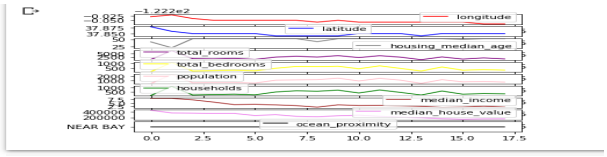


Fig. 5. Plot of the subplots

In next step I split the dataset using the sklearn library into training and testing where training the data is 70% and testing data is 30%.

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size= 0.3)

x_train_np = x_train.to_numpy()
y_train_np = y_train.to_numpy()

x_test_np = x_test.to_numpy()
y_test_np = y_test.to_numpy()
```

Fig. 6. Splitting the dataset

After the splitting of the data next will be performed is importing the some of the pytorch libraries which will be further used to generate the model. So, following is the code snippet for the libraries.

```
import torch
from torch.nn import Conv1d
from torch.nn import MaxPool1d
from torch.nn import Flatten
from torch.nn import Linear
from torch.nn.functional import relu
from torch.utils.data import DataLoader, TensorDataset
```

Fig. 7. pytorch libraries

In next step I applied CnnRegressor class which inherits torch.nn.Module and take the three positional arguments as a init method(constructor in other language) which are batch_size,inputs and outputs respectively. In this __init__ method inputs,outputs,max_pooling_layer, conv_layer, flatten_layer, linear_layer, output_layer are initialized. and then after I applied feed function which accepts input as an argument which generates the output. Following is the code snippet for the CnnRegressor class.

```
class CnnRegressor(torch.nn.Module):
  def __init__(self, batch_size, inputs, outputs):
    super(CnnRegressor, self).__init__()
    self.batch_size = batch_size
    self.inputs = inputs
    self.outputs = outputs
    self.input_layer = Conv1d(inputs, batch_size, 1)
    self.max_pooling_layer = MaxPool1d(1)
    self.conv_layer = Conv1d(batch_size, 128, 1)
    self.flatten_layer = Flatten()
    self.linear_layer = Linear(128, 128)
    self.output_layer = Linear(128, outputs)

  def feed(self, input):
    input = input.reshape((self.batch_size, self.inputs,1))
    output = relu(self.input_layer(input))
    output = self.max_pooling_layer(output)
    output = relu(self.conv_layer(output))
    output = self.flatten_layer(output)
    output = self.linear_layer(output)
    output = self.output_layer(output)
    return output
```

Fig. 8. Cnn regressor class

after initializing the some variable I imported the Adam optimizer for the Stochastic Optimization from the torch.optim library and also imported L1loss and then after also imported R2score from the ignite.contrib.metrics.regression.r2_score to calculate R2score of the model and for that I installed pytorch-ignite. Following is the code-snippet for the same.

```
from torch.optim import Adam
from torch.nn import L1loss
!pip install pytorch-ignite
from ignite.contrib.metrics.regression.r2_score import R2Score

Collecting pytorch-ignite
  Downloading https://files.pythonhosted.org/packages/35/55/41e8a995876fd2ade29bdba0c3efefa38e7d605cb353c70f3173
     |████████████████████████████████| 112kB 7.2MB/s
Requirement already satisfied: torch in /usr/local/lib/python3.6/dist-packages (from pytorch-ignite) (1.4.0)
Installing collected packages: pytorch-ignite
Successfully installed pytorch-ignite-0.3.0
```

Fig. 9. library for the optimization and some other task

In next step i initialized the batch size of 64 and also initialized Cnnregressor class where i also used cuda() function that implement the same function as CPU tensors, but they utilize GPUs for computation. code snippet and output for the same is given below.

```
[ ] batch_size = 64
    model = CnnRegressor(batch_size, X.shape[1], 1)
    model.cuda()

CnnRegressor(
  (input_layer): Conv1d(8, 64, kernel_size=(1,), stride=(1,))
  (max_pooling_layer): MaxPool1d(kernel_size=1, stride=1, padding=0, dilation=1, ceil_mode=False)
  (conv_layer): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
  (flatten_layer): Flatten()
  (linear_layer): Linear(in_features=128, out_features=128, bias=True)
  (output_layer): Linear(in_features=128, out_features=1, bias=True)
)
```

Fig. 10. initialization of the batch size and cuda() method

Then I performed the model_loss function which has 4 arguments which are model, dataset, train with default False value and optimizer.This function counts L1loss and R2Score error avg_loss, avg_score and returns those values to the calling function. Here I have used zero_grad() method which clears old gradients from the last step.In this function feed function is called which is the part of the CnnRegressor()

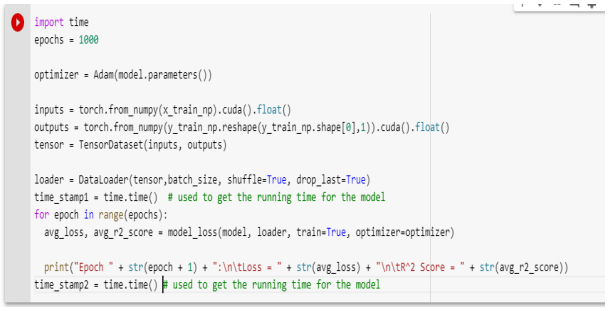
```
[ ] def model_loss(model, dataset, train = False, optimizer = None):
      performance = L1Loss()
      score_metric = R2Score()
      avg_loss = 0
      avg_score = 0
      count = 0
      for input, output in iter(dataset):
        predections = model.feed(input)
        loss = performance(predections, output)
        score_metric.update([predections, output])
        score = score_metric.compute()
        if(train):
          optimizer.zero_grad()
          loss.backward()
          optimizer.step()
        avg_loss += loss.item()
        avg_score += score
        count += 1
      return avg_loss / count, avg_score / count
```

Fig. 11. model_loss function

From now onwards there will be training of the model will be done

V. Training of the model

In training kernel size used for the model is 1 and batch size of the model is 64 which trains the model with Adam optimizer. I checked for the different types of the optimizer like Adam, R2prop, SGD(stochastic gradient descent) and many others also with the different learning rate but best accuracy i got with adom optimizer with default learning rate.For calculation purpose L1loss and R2Score packages are installed. After that to calculate L1loss and R2Score one method called model_loss is defined and these scores are stored in a counter and it is been returned in last.In my model epochs are set to 1000 and i got the output accuracy in training is around 72% and the time taken for the training the model is around 900 second. To get the time first i initialized time.time() variable above the loop and one other variable after the loop and then subtracted those two variable.



```
import time
epochs = 1000

optimizer = Adam(model.parameters())

inputs = torch.from_numpy(x_train_np).cuda().float()
outputs = torch.from_numpy(y_train_np.reshape(y_train_np.shape[0],1)).cuda().float()
tensor = TensorDataset(inputs, outputs)

loader = DataLoader(tensor,batch_size, shuffle=True, drop_last=True)
time_stamp1 = time.time()  # used to get the running time for the model
for epoch in range(epochs):
  avg_loss, avg_r2_score = model_loss(model, loader, train=True, optimizer=optimizer)

  print("Epoch " + str(epoch + 1) + ":\n\tLoss = " + str(avg_loss) + "\n\tR^2 Score = " + str(avg_r2_score))
time_stamp2 = time.time() # used to get the running time for the model
```

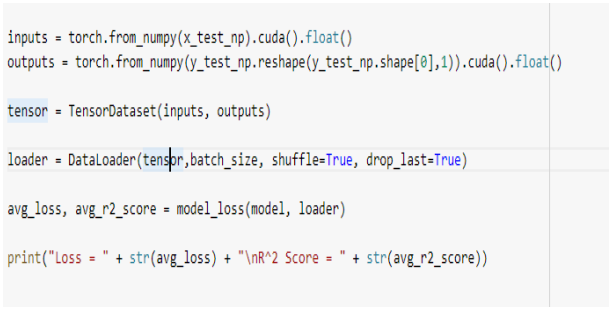
Fig. 12. Training the model

Then after i created model in pytorch with pth extension which is shown below.where i have given the filepath of directory and then download it.

```
print(f"Time taken for training the model: {time_stamp2-time_stamp1} second")
filepath = "/content/sample_data/kevin.pth"
torch.save(model,filepath)
model = torch.load(filepath)
```

Fig. 13. saving the model

Then after in testing i performed with same model with the 30% of remaining data to get the prediction where i got the accuracy around 69% to 70% so in that case i am not getting overfitting issues in model.



```
inputs = torch.from_numpy(x_test_np).cuda().float()
outputs = torch.from_numpy(y_test_np.reshape(y_test_np.shape[0],1)).cuda().float()

tensor = TensorDataset(inputs, outputs)

loader = DataLoader(tensor,batch_size, shuffle=True, drop_last=True)

avg_loss, avg_r2_score = model_loss(model, loader)

print("Loss = " + str(avg_loss) + "\nR^2 Score = " + str(avg_r2_score))
```

Fig. 14. testing the model

VI. Conclusion

I went through different different combination of the learning rate, epoch, optimizer, even i also added extra convolution layer to get the better accuracy but i got the better result with the adam optimizer with default learning rate which is 1e-3.

References

[1] https://pytorch.org/docs/stable/index.html
[2] https://www.wikipedia.org